REMARKS

1. The present invention, as claimed

It is conventional that when a process encounters a breakpoint, a debugger temporarily removes the breakpoint, an instruction for which the breakpoint was originally substituted is executed, and then the breakpoint is immediately replaced so that if the instruction is again encountered the breakpoint will fire. The present invention deals with a problem that arises when two threads encounter the same breakpoint and the debugger temporarily removes the breakpoint for one of the threads while processing of the breakpoint for the second thread is still pending. That is, when the processing of the breakpoint occurs for the second thread, it is problematic if the breakpoint has been temporarily removed because of the first thread. Page 5, lines 18 through 20. In such a case, the absent breakpoint is referred to in the present application as a "zombie" breakpoint and the problem is referred to as the "zombie breakpoint problem."

Claim 1 states that a breakpoint data structure is checked to determine if the data structure has an entry for a breakpoint known to a debugging process for a certain address where a breakpoint fired. Then, the claim goes on, there is a second step of verifying if a breakpoint condition continues to exist at the address where the breakpoint fired. As the claim states, this second step occurs if no entry is found by the first step. If the breakpoint does not exist, i.e., because it has been temporarily removed, the claim states that the breakpoint is identified as a zombie breakpoint. Independent claims 7 and 13 have similar language. This is in contrast to the conventional result, in which it is incorrectly concluded, due to the absence of the removed breakpoint, that the exception was not caused by a breakpoint.

2. Rejections under 35 USC 112, second paragraph

Claims 1-18 stand rejected under 35 USC 112, second paragraph. The Office action asserts that the independent claims 1, 7 and 13 are indefinite because i) there is no description of what the checking function was or did, ii) there is no description of how the determining is conducted, and iii) the determining is only an intended action. Applicant respectfully contends the rejections are improper for the following reasons.

Notwithstanding the following remarks, Applicant understands there are profound time pressures attending the examination process. Understandably, this sometimes leads to again be posed, "What else does it do when you touch a green widget by a magnet to cause an attraction of a certain force between the magnet and the widget if the widget is a certain metal?" The real issue should be whether Applicant has clearly stated something that is different and nonobvious with respect to the prior art, not whether Applicant has stated a certain amount of specific detail in the claims about how or what. Breadth of a claim is not to be equated with indefiniteness. MPEP 2173.04 (citing In re Miller, 441 F.2d 689, 169 USPQ 597 (CCPA 1971)).

3. Rejections under 35 USC 102(e) based on Bhattacarya

Claims 1-18 stand rejected under 35 USC 102(e) as being anticipated by U.S. patent 6,708,326 ("Bhattacarya"). Applicant respectfully contends the claims are patentably distinct, for the following reasons. To provide a context for the Bhattacarya passage cited in the Office action, consider the following teachings of Bhattacarya.

Bhattacarya primarily concerns improvement in speed of handling breakpoints. See Bhattacarya, col. 2, lines 18-29; col. 4, lines 22-33 (describing that breakpoint processing may be faster by avoiding the single-stepping conventionally done in the processing of breakpoints). Bhattacarya does mention a multi-thread problem of missed breakpoints. Bhattacarya, col. 2, lines 30-34. Bhattacarya teaches that this is dealt with by stopping all other threads when a first thread encounters a breakpoint, so that certain settings may be changed. Bhattacarya, col. 4, lines 12-21. But claim 1 in the present case states that a data structure is checked for an entry for a breakpoint known to a debugging process, and, if there is no entry, the method includes verifying if a breakpoint condition continues to exist at the address where the breakpoint fired. Claims 7 and 13 have similar language. Bhattacarya does not teach this.

Bhattacarya's method of handling breakpoints involves the following steps, which have conventionally been used in connection with putting breakpoint instructions into a debugee program instruction stream:

> (i) storing at respective addresses of said memory a sequence of processor instructions to be processed by the processor;
> (ii) replacing one of said processor instructions in said sequence with a break instruction, wherein in step (i) the one of said processor instructions was stored in a certain one of the addresses, and wherein the replacing includes storing the break instruction in the memory at the certain address in place of said one processor instruction;

breakpoint in the computer program may be hit by all jobs . . . regardless of whether the users that own the jobs are actively debugging the computer program . . . execution of the job that hit the breakpoint is suspended, and the encounter is reported to the user performing debugging. If the owner of the job is not performing the debugging, however, the job essentially appears hung or stalled to the owner until it is restarted by the user performing the debugging. As such, global breakpoints can be significantly disruptive in many computer environments." Bates, col. 2, lines 33-57.

The passage cited in the Office action continues, "if the hit control point is a service entry point." Bates, col. 8, line 39. It appears that Bates is making the point here that a hit control point might not be a breakpoint, in which case it is not the type of control point that Bates is concerned about. The cited passage continues, "control passes to block 94 to determine whether the current job is under debug." Bates, col. 8, lines 40-41. Presumably, this is because if a job is already under debug the debug service function does not need to be called again.

The cited passage continues, " If not, control passes to block 96 to determine whether the user ID that owns the current job is the same as that stored in the control point table for the service entry point. If so, the service entry function component is called in block 98." Bates, col. 8, lines 41-45. This provides a solution to the Bates problem description, " . . . all jobs that hit the global breakpoint, even those not under debug, will trigger the breakpoint. Furthermore . . . a breakpoint in the computer program may be hit by all jobs . . . regardless of whether the users that own the jobs are actively debugging the computer program." Bates, col. 2, lines 42-49. That is, if a job triggers a breakpoint but the job is not owned by a user who is debugging the job, Bates teaches that in this case the breakpoint should not call the debug service entry function. Thus, Bates is teaching when to ignore a breakpoint *that exists* but should not apply. This is different than, and does not teach, the claim 1 in the present case, which identifies an *absent* breakpoint (a zombie breakpoint), i.e., for which no entry is found in a data structure, by verifying no breakpoint condition continues to exist at the address where the breakpoint fired. Claims 7 and 13 have similar language.

PRIOR ART OF RECORD

Applicant has reviewed the prior art of record cited by but not relied upon by Examiner, and assert that the invention is patentably distinct.

REQUESTED ACTION

Applicant contends that the invention as claimed in accordance with amendments previously submitted is patentably distinct, and hereby requests that Examiner grant allowance and prompt passage of the application to issuance.

Respectfully submitted,

Anthony VS England

Anthony V. S. England
Attorney for Applicants
Registration No. 35,129
512-477-7165
a@aengland.com

cc: Applicant Initiated Interview Request